UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P O Box 1450
Alexandria, Virginia 22313-1450
www uspto gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/708,021	02/03/2004	Jonas Hogstrom	BORL/0217.00	2020

28653 7590 07/18/2008
JOHN A. SMART
201 LOS GATOS
SARATOGA RD, #161
LOS GATOS, CA 95030-5308

EXAMINER	
CHEN, QING	

ART UNIT	PAPER NUMBER
2191	

MAIL DATE	DELIVERY MODE
07/18/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	**Application No.** 10/708,021	**Applicant(s)** HOGSTROM ET AL.	
	Examiner Qing Chen	**Art Unit** 2191	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *18 April 2008*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-63* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-63* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)

2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.

4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.

5) ☐ Notice of Informal Patent Application

6) ☐ Other: _____.

DETAILED ACTION

1. This Office action is in response to the amendment filed on April 18, 2008.

2. **Claims 1-63** are pending.

3. **Claims 1, 7, 21, 23, 29, 43, 48, and 62** have been amended.

4. The objections to Claims 1 and 23 are withdrawn in view of Applicant's amendments to the claims.

5. The 35 U.S.C. § 112, second paragraph, rejections of Claims 21 and 62 are withdrawn in view of Applicant's amendments to the claims.

Response to Amendment

Claim Rejections - 35 USC § 103

6. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

7. **Claims 1-7, 15-29, 37-48, and 56-63** are rejected under 35 U.S.C. 103(a) as being unpatentable over **US 6,199,195 (hereinafter "Goodwin")** in view of **US 7,000,219 (hereinafter "Barrett")**.

As per **Claim 1**, Goodwin discloses:

- creating a model describing business objects and rules of the application *(see Column 6: 37-44, "The system and method will also allow developers to generate objects based on a framework of services they author by composing services based on the object templates into objects that support the composed behaviors and methods. This is accomplished through the code generator 210, which interfaces with the unified models 206, which are expressed in a unified modeling language, such as Unified Modeling Language (UML).")*;

- creating source code for the application *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ...")*;

- compiling the source code into an executable application *(see Column 14: 34-40, "Outputs from the code generator 404 include Interface Definition Language (IDL) files 422 (*.idl), which are processed by an idl-to-Java compiler, e.g., Visibroker's IDL2JAVA, for the generation of CORBA ORB services ...")*; and

- running the executable application on a target computer in conjunction with a run-time framework that provides services to the executable application *(see Column 15: 45-65, "The data server 332 operates at run time ..." and "When deployed within a client application, the data server 332 launches, starts, manages and controls execution of a set of services.")*.

However, Goodwin does not disclose:

- representing the model within the source code itself; and

- while the executable application is running, reconstructing the model from the executable application and making it available to the run-time framework.

Barrett discloses:

- representing the model within the source code itself *(see Column 5: 48-52, "In the requirements and specification phases 1 and 2 the system design is modelled. In the preferred embodiment, this model comprises a series of UML diagrams. The model is developed by the UML tool 15 and recorded in a file in an XMI/XML form.")*; and

- while the executable application is running, reconstructing the model from the executable application and making it available to the run-time framework *(see Column 5: 37-44, "During execution of a system, the RAS 10 maintains the meta-model associated with the executing system. During this phase, the meta model remains accessible to the UML tool. Changes made to the meta model have immediate effect on the executing system by addition, removal and/or replacement of components, and changes to the structure of the bindings between instances according to changes to the meta model.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Barrett into the teaching of Goodwin to include representing the model within the source code itself; and while the executable application is running, reconstructing the model from the executable application and making it available to the run-time framework. The modification would be obvious because one of ordinary skill in the art would be motivated to maintain a "flexible" software system *(see Barrett – Column 1: 11-16).*

As per **Claim 2**, the rejection of **Claim 1** is incorporated; and Goodwin further discloses:

- wherein the model comprises a Unified Modeling Language (UML) model *(see Column 6: 37-44, "... the unified models 206, which are expressed in a unified modeling language, such as Unified Modeling Language (UML).")*.

As per **Claim 3**, the rejection of **Claim 1** is incorporated; and Goodwin further discloses:

- wherein the source code is created using a programming language *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ...")*.

As per **Claim 4**, the rejection of **Claim 3** is incorporated; and Goodwin further discloses:

- wherein the programming language is an object oriented programming language *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ...")*.

As per **Claim 5**, the rejection of **Claim 3** is incorporated; and Goodwin further discloses:

- wherein the programming language is one that supports reflection technique, thereby allowing reconstruction of the model at run-time from the executable application *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ..." It is inherent that Java™ supports the reflection technique.)*.

As per **Claim 6**, the rejection of **Claim 1** is incorporated; and Goodwin further discloses:

- wherein the reconstructed model is employed at run-time to support services that the run-time framework provides to the executable application *(see Column 16: 65-67 through Column 17: 1, "The first step in preparing to use the data server 338 (i.e., preparing the data server 338 for run time operation) consists of developing a unified model of a business application and storing that model in the schema repository 314.").*

As per **Claim 7**, the rejection of **Claim 1** is incorporated; however, Goodwin does not disclose:

- using reflection, reading metadata associated with the executable application to create a graph of code elements; and

- spanning the graph for re-creating the model based on code elements encountered.

Barrett discloses:

- reading metadata to create a graph of code elements *(see Column 7: 35-38, "The XMI adapter 30 creates a hierarchy of objects from an XMI file, and allows modifications, and the generation of an XMI file. This object set forms a hierarchical graph of connected objects, called the meta model.")*; and

- spanning the graph for re-creating the model based on code elements encountered *(see Figure 10; Column 8: 46-48, "FIG. 10 illustrates the Document Object Model (DOM) representation of the example XMI model.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Barrett into the teaching of Goodwin to include reading metadata to create a graph of code elements; and spanning the graph for re-

creating the model based on code elements encountered. The modification would be obvious because one of ordinary skill in the art would be motivated to access or track all elements of the UML.

Official Notice is taken that it is old and well-known within the computing art to read metadata in an application using reflection. Reflective programming is a programming paradigm, used as an extension to the object-oriented programming paradigm, to add self-optimization to application programs, and to improve their flexibility. Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to include reading metadata in an application using reflection. The modification would be obvious because one of ordinary skill in the art would be motivated to gather metadata from the application during runtime.

As per **Claim 15**, the rejection of **Claim 1** is incorporated; and Goodwin further discloses:

- wherein the reconstructed model is stored in a cache memory available to the run-time framework *(see Column 5: 45-47, "The illustrated system 100 has a processor 102 coupled to a memory 104 ...").*

As per **Claim 16**, the rejection of **Claim 1** is incorporated; and Goodwin further discloses:

- wherein the model is initially created using a modeling tool, and wherein the source code is compiled using a compiler *(see Column 8: 44-58, "Shown are a number of modeling tools 302, 304, 306 both data modeling 302 and object modeling 304, 306, defining data within a*

database 308 or defining objects and relating these objects to the data within the database 308."; Column 14: 34-40, "Outputs from the code generator 404 include Interface Definition Language (IDL) files 422 (.idl), which are processed by an idl-to-Java compiler, e.g., Visibroker's IDL2JAVA, for the generation of CORBA ORB services ...").*

As per **Claim 17**, the rejection of **Claim 1** is incorporated; however, Goodwin does not disclose:

- representing information of the model in source code as language constructs.

Barrett discloses:

- representing information of the model in source code as language constructs *(see Column 8: 30-44).*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Barrett into the teaching of Goodwin to include representing information of the model in source code as language constructs. The modification would be obvious because one of ordinary skill in the art would be motivated to describe the UML in code form.

As per **Claim 18**, the rejection of **Claim 1** is incorporated; however, Goodwin does not disclose:

- representing information of the model in source code as attributes.

Barrett discloses:

- representing information of the model in source code as attributes *(see Column 8: 30-44)*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Barrett into the teaching of Goodwin to include representing information of the model in source code as attributes. The modification would be obvious because one of ordinary skill in the art would be motivated to describe the UML in code form.

As per **Claim 19**, the rejection of **Claim 18** is incorporated; however, Goodwin does not disclose:

- wherein attributes comprise specifiers to structural code elements.

Barrett discloses:

- wherein attributes comprise specifiers to structural code elements *(see Column 8: 30-44)*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Barrett into the teaching of Goodwin to include wherein attributes comprise specifiers to structural code elements. The modification would be obvious because one of ordinary skill in the art would be motivated to describe the UML in code form.

As per **Claim 20**, the rejection of **Claim 1** is incorporated; however, Goodwin does not disclose:

- representing information of the model in code artifacts that exist expressly for carrying model information in source code.

Barrett discloses:

- representing information of the model in code artifacts that exist expressly for carrying model information in source code *(see Column 8: 30-44).*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Barrett into the teaching of Goodwin to include representing information of the model in code artifacts that exist expressly for carrying model information in source code. The modification would be obvious because one of ordinary skill in the art would be motivated to describe the UML in code form.

As per **Claim 21**, the rejection of **Claim 1** is incorporated; and Goodwin further discloses:

- a computer-readable medium storing processor-executable instructions for performing the method of claim 1 *(see Column 5: 45-47, "The illustrated system 100 has a processor 102 coupled to a memory 104 ...").*

As per **Claim 22**, the rejection of **Claim 1** is incorporated; and Goodwin further discloses:

- a downloadable set of processor-executable instructions for performing the method of claim 1 stored on a computer-readable medium *(see Column 9: 52-57, "The code generator 330 writes source code objects to a directory and returns a uniform resource locator (URL)*

identified to a client application 338. An operator of the client application (338) is then required to go to the location identified by the uniform resource locator and download the generated code.").

As per **Claim 23**, Goodwin discloses:

- a computer system having a processor and memory *(see Figure 1: 100, 102, and 104)*;

- a modeling tool for creating a model describing business objects and rules of the application *(see Column 6: 37-44, "The system and method will also allow developers to generate objects based on a framework of services they author by composing services based on the object templates into objects that support the composed behaviors and methods. This is accomplished through the code generator 210, which interfaces with the unified models 206, which are expressed in a unified modeling language, such as Unified Modeling Language (UML).")*;

- a module for creating source code for the application *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ...")*; and

- a compiler for compiling the source code into an executable application *(see Column 14: 34-40, "Outputs from the code generator 404 include Interface Definition Language (IDL) files 422 (*.idl), which are processed by an idl-to-Java compiler, e.g., Visibroker's IDL2JAVA, for the generation of CORBA ORB services ...").*

However, Goodwin does not disclose:

- a module for representing the model within the source code itself; and

- a run-time framework that is able to reconstruct the model from the executable application and use it for providing services.

Barrett discloses:

- a module for representing the model within the source code itself *(see Column 5: 48-52, "In the requirements and specification phases 1 and 2 the system design is modelled. In the preferred embodiment, this model comprises a series of UML diagrams. The model is developed by the UML tool 15 and recorded in a file in an XMI/XML form.")*; and

- a run-time framework that is able to reconstruct the model from the executable application and use it for providing services *(see Column 5: 37-44, "During execution of a system, the RAS 10 maintains the meta-model associated with the executing system. During this phase, the meta model remains accessible to the UML tool. Changes made to the meta model have immediate effect on the executing system by addition, removal and/or replacement of components, and changes to the structure of the bindings between instances according to changes to the meta model.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Barrett into the teaching of Goodwin to include a module for representing the model within the source code itself; and a run-time framework that is able to reconstruct the model from the executable application and use it for providing services. The modification would be obvious because one of ordinary skill in the art would be motivated to maintain a "flexible" software system *(see Barrett – Column 1: 11-16).*

As per **Claim 24**, the rejection of **Claim 23** is incorporated; and Goodwin further discloses:

- wherein the model comprises a Unified Modeling Language (UML) model *(see Column 6: 37-44, "... the unified models 206, which are expressed in a unified modeling language, such as Unified Modeling Language (UML).").*

As per **Claim 25,** the rejection of **Claim 23** is incorporated; and Goodwin further discloses:

- wherein the source code is created using a programming language *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ...").*

As per **Claim 26,** the rejection of **Claim 25** is incorporated; and Goodwin further discloses:

- wherein the programming language is an object oriented programming language *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ...").*

As per **Claim 27**, the rejection of **Claim 25** is incorporated; and Goodwin further discloses:

- wherein the programming language is one that supports reflection technique, thereby allowing reconstruction of the model at run-time from the executable application *(see Column*

13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ..." It is inherent that Java™ supports the reflection technique.).

As per **Claim 28**, the rejection of **Claim 23** is incorporated; and Goodwin further discloses:

- wherein the reconstructed model is employed at run-time to support services that the run-time framework provides to the executable application *(see Column 16: 65-67 through Column 17: 1, "The first step in preparing to use the data server 338 (i.e., preparing the data server 338 for run time operation) consists of developing a unified model of a business application and storing that model in the schema repository 314.").*

As per **Claim 29**, the rejection of **Claim 23** is incorporated; however, Goodwin does not disclose:

- wherein the run-time framework includes submodules for reading metadata associated with the executable application to create a graph of code elements using reflection, and for spanning the graph for re-creating the model based on code elements encountered.

Barrett discloses:

- wherein the run-time framework includes submodules for reading metadata to create a graph of code elements, and for spanning the graph for re-creating the model based on code elements encountered *(see Figure 10; Column 7: 35-38, "The XMI adapter 30 creates a hierarchy of objects from an XMI file, and allows modifications, and the generation of an XMI file. This object set forms a hierarchical graph of connected objects, called the meta model.";*

Column 8: 46-48, "FIG. 10 illustrates the Document Object Model (DOM) representation of the example XMI model.").

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Barrett into the teaching of Goodwin to include wherein the run-time framework includes submodules for reading metadata to create a graph of code elements, and for spanning the graph for re-creating the model based on code elements encountered. The modification would be obvious because one of ordinary skill in the art would be motivated to access or track all elements of the UML.

Official Notice is taken that it is old and well-known within the computing art to read metadata in an application using reflection. Reflective programming is a programming paradigm, used as an extension to the object-oriented programming paradigm, to add self-optimization to application programs, and to improve their flexibility. Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to include reading metadata in an application using reflection. The modification would be obvious because one of ordinary skill in the art would be motivated to gather metadata from the application during runtime.

As per **Claim 37**, the rejection of **Claim 23** is incorporated; and Goodwin further discloses:

- wherein the reconstructed model is stored in a cache memory available to the run-time framework *(see Column 5: 45-47, "The illustrated system 100 has a processor 102 coupled to a memory 104 ...").*

As per **Claim 38**, the rejection of **Claim 23** is incorporated; and Goodwin further discloses:

- wherein the model is initially created using a UML modeling tool *(see Column 8: 44-53, "Shown are a number of modeling tools 302, 304, 306 both data modeling 302 and object modeling 304, 306, defining data within a database 308 or defining objects and relating these objects to the data within the database 308." and "... a plurality of model adapters 310 for defining a translation of the logical models of the modeling tools 302, 304, 406 into unified models, expressed in a unified modeling language, such as Unified Modeling Language (UML).")*.

However, Goodwin and Barrett do not disclose:

- wherein the source code is compiled using a C# compiler.

Official Notice is taken that it is old and well-known within the computing art to include compiling source code using a C# compiler. A compiler is an essential software component of an Integrated Development Environment (IDE) used by computer programmers to develop software, such as C# applications. Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to include wherein the source code is compiled using a C# compiler. The modification would be obvious because one of ordinary skill in the art would be motivated to execute applications written in C#.

As per **Claim 39**, the rejection of **Claim 23** is incorporated; however, Goodwin does not disclose:

- wherein the module for creating source code is able to represent information of the model in source code as language constructs.

Barrett discloses:

- wherein the module for creating source code is able to represent information of the model in source code as language constructs *(see Column 8: 30-44)*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Barrett into the teaching of Goodwin to include wherein the module for creating source code is able to represent information of the model in source code as language constructs. The modification would be obvious because one of ordinary skill in the art would be motivated to describe the UML in code form.

As per **Claim 40,** the rejection of **Claim 23** is incorporated; however, Goodwin does not disclose:

- wherein the module for creating source code is able to represent information of the model in source code as attributes.

Barrett discloses:

- wherein the module for creating source code is able to represent information of the model in source code as attributes *(see Column 8: 30-44)*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Barrett into the teaching of Goodwin to include wherein the module for creating source code is able to represent information of the

model in source code as attributes. The modification would be obvious because one of ordinary skill in the art would be motivated to describe the UML in code form.

As per **Claim 41**, the rejection of **Claim 40** is incorporated; however, Goodwin does not disclose:

- wherein attributes comprise specifiers to structural code elements.

Barrett discloses:

- wherein attributes comprise specifiers to structural code elements *(see Column 8: 30-44)*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Barrett into the teaching of Goodwin to include wherein attributes comprise specifiers to structural code elements. The modification would be obvious because one of ordinary skill in the art would be motivated to describe the UML in code form.

As per **Claim 42**, the rejection of **Claim 23** is incorporated; however, Goodwin does not disclose:

- wherein the module for creating source code is able to represent information of the model in code artifacts that exist expressly for carrying model information in source code.

Barrett discloses:

- wherein the module for creating source code is able to represent information of the model in code artifacts that exist expressly for carrying model information in source code *(see Column 8: 30-44)*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Barrett into the teaching of Goodwin to include wherein the module for creating source code is able to represent information of the model in code artifacts that exist expressly for carrying model information in source code. The modification would be obvious because one of ordinary skill in the art would be motivated to describe the UML in code form.

As per **Claim 43**, Goodwin discloses:

- creating a model for developing an application using Unified Modeling Language (UML) technique *(see Column 6: 37-44, "The system and method will also allow developers to generate objects based on a framework of services they author by composing services based on the object templates into objects that support the composed behaviors and methods. This is accomplished through the code generator 210, which interfaces with the unified models 206, which are expressed in a unified modeling language, such as Unified Modeling Language (UML).")*;

- generating source code to implement the model *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ...")*; and

- compiling the amended source code into an executable application and running the executable application on the computer system *(see Column 14: 34-40, "Outputs from the code*

generator 404 include Interface Definition Language (IDL) files 422 (.idl), which are processed by an idl-to-Java compiler, e.g., Visibroker's IDL2JAVA, for the generation of CORBA ORB services ..."; Column 15: 45-65, "The data server 332 operates at run time ..." and "When deployed within a client application, the data server 332 launches, starts, manages and controls execution of a set of services.").*

However, Goodwin does not disclose:

- amending the source code for storing model information in the source code;
- reconstructing the model from the executable application; and
- making the reconstructed model available for supporting operation of the executable application, including rendering the reconstructed model for display.

Barrett discloses:

- amending the source code for storing model information in the source code *(see Column 5: 48-52, "In the requirements and specification phases 1 and 2 the system design is modelled. In the preferred embodiment, this model comprises a series of UML diagrams. The model is developed by the UML tool 15 and recorded in a file in an XMI/XML form.")*;
- reconstructing the model from the executable application *(see Column 5: 37-44, "During execution of a system, the RAS 10 maintains the meta-model associated with the executing system. During this phase, the meta model remains accessible to the UML tool. Changes made to the meta model have immediate effect on the executing system by addition, removal and/or replacement of components, and changes to the structure of the bindings between instances according to changes to the meta model.")*; and

- making the reconstructed model available for supporting operation of the executable application, including rendering the reconstructed model for display *(see Column 5: 37-44, "During execution of a system, the RAS 10 maintains the meta-model associated with the executing system. During this phase, the meta model remains accessible to the UML tool. Changes made to the meta model have immediate effect on the executing system by addition, removal and/or replacement of components, and changes to the structure of the bindings between instances according to changes to the meta model.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Barrett into the teaching of Goodwin to include amending the source code for storing model information in the source code; reconstructing the model from the executable application; and making the reconstructed model available for supporting operation of the executable application, including rendering the reconstructed model for display. The modification would be obvious because one of ordinary skill in the art would be motivated to maintain a "flexible" software system *(see Barrett – Column 1: 11-16)*.

As per **Claim 44**, the rejection of **Claim 43** is incorporated; and Goodwin further discloses:

- wherein the source code is implemented using a programming language *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ...")*.

As per **Claim 45**, the rejection of **Claim 44** is incorporated; and Goodwin further discloses:

- wherein the programming language is an object oriented programming language *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ...").*

As per **Claim 46**, the rejection of **Claim 45** is incorporated; and Goodwin further discloses:

- wherein the object oriented programming language is one that supports reflection technique, thereby allowing reconstruction of the model from the executable application *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ..." It is inherent that Java™ supports the reflection technique.).*

As per **Claim 47**, the rejection of **Claim 43** is incorporated; and Goodwin further discloses:

- wherein the reconstructed model is employed by a run-time framework to provide services to the executable application *(see Column 16: 65-67 through Column 17: 1, "The first step in preparing to use the data server 338 (i.e., preparing the data server 338 for run time operation) consists of developing a unified model of a business application and storing that model in the schema repository 314.").*

As per **Claim 48**, the rejection of **Claim 43** is incorporated; however, Goodwin does not disclose:

- using reflection, reading metadata associated with the executable application to create a graph of code elements; and
- spanning the graph for re-creating the model based on code elements encountered.

Barrett discloses:

- reading metadata to create a graph of code elements *(see Column 7: 35-38, "The XMI adapter 30 creates a hierarchy of objects from an XMI file, and allows modifications, and the generation of an XMI file. This object set forms a hierarchical graph of connected objects, called the meta model.")*; and
- spanning the graph for re-creating the model based on code elements encountered *(see Figure 10; Column 8: 46-48, "FIG. 10 illustrates the Document Object Model (DOM) representation of the example XMI model.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Barrett into the teaching of Goodwin to include reading metadata to create a graph of code elements; and spanning the graph for re-creating the model based on code elements encountered. The modification would be obvious because one of ordinary skill in the art would be motivated to access or track all elements of the UML.

Official Notice is taken that it is old and well-known within the computing art to read metadata in an application using reflection. Reflective programming is a programming paradigm, used as an extension to the object-oriented programming paradigm, to add self-optimization to

application programs, and to improve their flexibility. Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to include reading metadata in an application using reflection. The modification would be obvious because one of ordinary skill in the art would be motivated to gather metadata from the application during runtime.

As per **Claim 56,** the rejection of **Claim 43** is incorporated; and Goodwin further discloses:

- wherein the reconstructed model is stored in a cache memory *(see Column 5: 45-47, "The illustrated system 100 has a processor 102 coupled to a memory 104 ...").*

As per **Claim 57,** the rejection of **Claim 43** is incorporated; and Goodwin further discloses:

- wherein the model is initially created using a modeling tool, and wherein the amended source code is compiled using a compiler *(see Column 8: 44-58, "Shown are a number of modeling tools 302, 304, 306 both data modeling 302 and object modeling 304, 306, defining data within a database 308 or defining objects and relating these objects to the data within the database 308."; Column 14: 34-40, "Outputs from the code generator 404 include Interface Definition Language (IDL) files 422 (*.idl), which are processed by an idl-to-Java compiler, e.g., Visibroker's IDL2JAVA, for the generation of CORBA ORB services ...").*

As per **Claim 58**, the rejection of **Claim 43** is incorporated; however, Goodwin does not disclose:

- representing information of the model in source code as language constructs.

Barrett discloses:

- representing information of the model in source code as language constructs *(see Column 8: 30-44)*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Barrett into the teaching of Goodwin to include representing information of the model in source code as language constructs. The modification would be obvious because one of ordinary skill in the art would be motivated to describe the UML in code form.

As per **Claim 59**, the rejection of **Claim 43** is incorporated; however, Goodwin does not disclose:

- representing information of the model in source code as attributes.

Barrett discloses:

- representing information of the model in source code as attributes *(see Column 8: 30-44)*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Barrett into the teaching of Goodwin to include representing information of the model in source code as attributes. The modification would be obvious because one of ordinary skill in the art would be motivated to describe the UML in code form.

As per **Claim 60**, the rejection of **Claim 59** is incorporated; however, Goodwin does not disclose:

- wherein attributes comprise specifiers to structural code elements.

Barrett discloses:

- wherein attributes comprise specifiers to structural code elements *(see Column 8: 30-44)*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Barrett into the teaching of Goodwin to include wherein attributes comprise specifiers to structural code elements. The modification would be obvious because one of ordinary skill in the art would be motivated to describe the UML in code form.

As per **Claim 61**, the rejection of **Claim 43** is incorporated; however, Goodwin does not disclose:

- representing information of the model in code artifacts that exist expressly for carrying model information in source code.

Barrett discloses:

- representing information of the model in code artifacts that exist expressly for carrying model information in source code *(see Column 8: 30-44)*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Barrett into the teaching of Goodwin to include representing information of the model in code artifacts that exist expressly for carrying

model information in source code. The modification would be obvious because one of ordinary skill in the art would be motivated to describe the UML in code form.

As per **Claim 62**, the rejection of **Claim 43** is incorporated; and Goodwin further discloses:

- a computer-readable medium storing processor-executable instructions for performing the method of claim 43 *(see Column 5: 45-47, "The illustrated system 100 has a processor 102 coupled to a memory 104 ...")*.

As per **Claim 63**, the rejection of **Claim 43** is incorporated; and Goodwin further discloses:

- a downloadable set of processor-executable instructions for performing the method of claim 43 stored on a computer-readable medium *(see Column 9: 52-57, "The code generator 330 writes source code objects to a directory and returns a uniform resource locator (URL) identified to a client application 338. An operator of the client application (338) is then required to go to the location identified by the uniform resource locator and download the generated code.")*.

8. **Claims 8, 10, 11, 30, 32, 33, 49, 51, and 52** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Goodwin** in view of **Barrett** as applied to Claims 1, 7, 23, 29, 43, and 48 above, and further in view of **US 6,560,769 (hereinafter "Moore")**.

As per **Claim 8**, the rejection of **Claim 7** is incorporated; however, Goodwin and Barrett do not disclose:

- as each code element is encountered, reconstructing a corresponding portion of the model.

Moore discloses:

- as each code element is encountered, reconstructing a corresponding portion of the model *(see Column 4: 41-50, "Referring now to FIG. 4 a flow chart illustrates the process for representing a JAVA file in UML. The process begins with a start bubble 40 followed by a step of parsing the JAVA source code to be represented in UML (block 41).").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Moore into the teaching of Goodwin to include as each code element is encountered, reconstructing a corresponding portion of the model. The modification would be obvious because one of ordinary skill in the art would be motivated to allow JAVA programmers to diagram JAVA code without having to create a separate model *(see Moore – Column 1: 43-46).*

As per **Claim 10**, the rejection of **Claim 1** is incorporated; however, Goodwin and Barrett do not disclose:

- detecting a class having a package element; and
- creating a corresponding Unified Modeling Language (UML) package for the reconstructed model.

Moore discloses:

- detecting a class having a package element *(see Column 4: 41-50, "Referring now to FIG. 4 a flow chart illustrates the process for representing a JAVA file in UML. The process begins with a start bubble 40 followed by a step of parsing the JAVA source code to be represented in UML (block 41)." and "After this, a UML package representing the JAVA package statement is created (block 44) ... ")*; and

- creating a corresponding Unified Modeling Language (UML) package for the reconstructed model *(see Column 4: 41-50, "Referring now to FIG. 4 a flow chart illustrates the process for representing a JAVA file in UML. The process begins with a start bubble 40 followed by a step of parsing the JAVA source code to be represented in UML (block 41)." and "After this, a UML package representing the JAVA package statement is created (block 44) ... ").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Moore into the teaching of Goodwin to include detecting a class having a package element; and creating a corresponding Unified Modeling Language (UML) package for the reconstructed model. The modification would be obvious because one of ordinary skill in the art would be motivated to allow JAVA programmers to diagram JAVA code without having to create a separate model *(see Moore – Column 1: 43-46)*.

As per **Claim 11**, the rejection of **Claim 10** is incorporated; however, Goodwin and Barrett do not disclose:

- detecting an attribute specifying that a class belongs to the UML package; and
- specifying in the reconstructed model that the class belongs to that UML package.

Moore discloses:

- detecting an attribute specifying that a class belongs to the UML package *(see Column 4: 51-55, "Following the above, an inquiry is made as to whether or not there is a JAVA class not represented in UML (diamond 46). If the answer to this inquiry is yes, then a UML class representing the JAVA class is created (block 47) followed by a return back to the diamond 46 for the next class, if any.")*; and

- specifying in the reconstructed model that the class belongs to that UML package *(see Column 4: 51-55, "Following the above, an inquiry is made as to whether or not there is a JAVA class not represented in UML (diamond 46). If the answer to this inquiry is yes, then a UML class representing the JAVA class is created (block 47) followed by a return back to the diamond 46 for the next class, if any.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Moore into the teaching of Goodwin to include detecting an attribute specifying that a class belongs to the UML package; and specifying in the reconstructed model that the class belongs to that UML package. The modification would be obvious because one of ordinary skill in the art would be motivated to allow JAVA programmers to diagram JAVA code without having to create a separate model *(see Moore – Column 1: 43-46).*

As per **Claim 30**, the rejection of **Claim 29** is incorporated; however, Goodwin and Barrett do not disclose:

- wherein the submodule for spanning is able to reconstruct portions of the model based on corresponding code elements encountered in the executable application.

Moore discloses:

- wherein the submodule for spanning is able to reconstruct portions of the model based on corresponding code elements encountered in the executable application *(see Column 4: 41-50, "Referring now to FIG. 4 a flow chart illustrates the process for representing a JAVA file in UML. The process begins with a start bubble 40 followed by a step of parsing the JAVA source code to be represented in UML (block 41).")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Moore into the teaching of Goodwin to include wherein the submodule for spanning is able to reconstruct portions of the model based on corresponding code elements encountered in the executable application. The modification would be obvious because one of ordinary skill in the art would be motivated to allow JAVA programmers to diagram JAVA code without having to create a separate model *(see Moore – Column 1: 43-46)*.

As per **Claim 32**, the rejection of **Claim 23** is incorporated; however, Goodwin and Barrett do not disclose:

- wherein the run-time framework includes submodules for detecting a class having a package element, and for creating a corresponding Unified Modeling Language (UML) package for the reconstructed model.

Moore discloses:

- wherein the run-time framework includes submodules for detecting a class having a package element, and for creating a corresponding Unified Modeling Language (UML) package for the reconstructed model *(see Column 4: 41-50, "Referring now to FIG. 4 a flow chart illustrates the process for representing a JAVA file in UML. The process begins with a start bubble 40 followed by a step of parsing the JAVA source code to be represented in UML (block 41)." and "After this, a UML package representing the JAVA package statement is created (block 44) ...")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Moore into the teaching of Goodwin to include wherein the run-time framework includes submodules for detecting a class having a package element, and for creating a corresponding Unified Modeling Language (UML) package for the reconstructed model. The modification would be obvious because one of ordinary skill in the art would be motivated to allow JAVA programmers to diagram JAVA code without having to create a separate model *(see Moore – Column 1: 43-46)*.

As per **Claim 33**, the rejection of **Claim 32** is incorporated; however, Goodwin and Barrett do not disclose:

- a module for detecting an attribute specifying that a class belongs to the UML package, and for specifying in the reconstructed model that the class belongs to that UML package.

Moore discloses:

- a module for detecting an attribute specifying that a class belongs to the UML package, and for specifying in the reconstructed model that the class belongs to that UML package *(see Column 4: 51-55, "Following the above, an inquiry is made as to whether or not there is a JAVA class not represented in UML (diamond 46). If the answer to this inquiry is yes, then a UML class representing the JAVA class is created (block 47) followed by a return back to the diamond 46 for the next class, if any.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Moore into the teaching of Goodwin to include a module for detecting an attribute specifying that a class belongs to the UML package, and for specifying in the reconstructed model that the class belongs to that UML package. The modification would be obvious because one of ordinary skill in the art would be motivated to allow JAVA programmers to diagram JAVA code without having to create a separate model *(see Moore – Column 1: 43-46)*.

As per **Claim 49**, the rejection of **Claim 48** is incorporated; however, Goodwin and Barrett do not disclose:

- as each code element is encountered, reconstructing a corresponding portion of the model.

Moore discloses:

- as each code element is encountered, reconstructing a corresponding portion of the model *(see Column 4: 41-50, "Referring now to FIG. 4 a flow chart illustrates the process for*

representing a JAVA file in UML. The process begins with a start bubble 40 followed by a step of parsing the JAVA source code to be represented in UML (block 41).").

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Moore into the teaching of Goodwin to include as each code element is encountered, reconstructing a corresponding portion of the model. The modification would be obvious because one of ordinary skill in the art would be motivated to allow JAVA programmers to diagram JAVA code without having to create a separate model *(see Moore – Column 1: 43-46).*

As per **Claim 51,** the rejection of **Claim 43** is incorporated; however, Goodwin and Barrett do not disclose:

- detecting a class having a package element; and
- creating a corresponding Unified Modeling Language (UML) package for the reconstructed model.

Moore discloses:

- detecting a class having a package element *(see Column 4: 41-50, "Referring now to FIG. 4 a flow chart illustrates the process for representing a JAVA file in UML. The process begins with a start bubble 40 followed by a step of parsing the JAVA source code to be represented in UML (block 41)." and "After this, a UML package representing the JAVA package statement is created (block 44) ...")*; and
- creating a corresponding Unified Modeling Language (UML) package for the reconstructed model *(see Column 4: 41-50, "Referring now to FIG. 4 a flow chart illustrates the*

process for representing a JAVA file in UML. The process begins with a start bubble 40 followed by a step of parsing the JAVA source code to be represented in UML (block 41)." and "After this, a UML package representing the JAVA package statement is created (block 44) ...").

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Moore into the teaching of Goodwin to include detecting a class having a package element; and creating a corresponding Unified Modeling Language (UML) package for the reconstructed model. The modification would be obvious because one of ordinary skill in the art would be motivated to allow JAVA programmers to diagram JAVA code without having to create a separate model *(see Moore – Column 1: 43-46).*

As per **Claim 52,** the rejection of **Claim 51** is incorporated; however, Goodwin and Barrett do not disclose:

- detecting an attribute specifying that a class belongs to the UML package; and
- specifying in the reconstructed model that the class belongs to that UML package.

Moore discloses:

- detecting an attribute specifying that a class belongs to the UML package *(see Column 4: 51-55, "Following the above, an inquiry is made as to whether or not there is a JAVA class not represented in UML (diamond 46). If the answer to this inquiry is yes, then a UML class representing the JAVA class is created (block 47) followed by a return back to the diamond 46 for the next class, if any.")*; and

- specifying in the reconstructed model that the class belongs to that UML package *(see Column 4: 51-55, "Following the above, an inquiry is made as to whether or not there is a JAVA class not represented in UML (diamond 46). If the answer to this inquiry is yes, then a UML class representing the JAVA class is created (block 47) followed by a return back to the diamond 46 for the next class, if any.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Moore into the teaching of Goodwin to include detecting an attribute specifying that a class belongs to the UML package; and specifying in the reconstructed model that the class belongs to that UML package. The modification would be obvious because one of ordinary skill in the art would be motivated to allow JAVA programmers to diagram JAVA code without having to create a separate model *(see Moore – Column 1: 43-46)*.

9. **Claims 9, 31, and 50** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Goodwin** in view of **Barrett** as applied to Claims 7, 29, and 48 above, and further in view of **US 2004/0044990 (hereinafter "Schloegel")**.

As per **Claim 9**, the rejection of **Claim 7** is incorporated; however, Goodwin and Barrett do not disclose:

- wherein the spanning step includes traversing the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques.

Schloegel discloses:

- wherein the spanning step includes traversing the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques *(see Paragraph [0033], "... the order of entity traversal during code generation could be random, or ordered by type, or sorted by name, or filtered so that only entities with a specific property are traversed, or the graph could be traversed in various other ways such as depth-first traversal.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Schloegel into the teaching of Goodwin to include wherein the spanning step includes traversing the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques. The modification would be obvious because one of ordinary skill in the art would be motivated to provide a framework under which it is possible to reason and/or prove qualities about the generated code *(see Schloegel – Paragraph [0034]).*

As per **Claim 31**, the rejection of **Claim 29** is incorporated; however, Goodwin and Barrett do not disclose:

- wherein the submodule for spanning is able to traverse the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques.

Schloegel discloses:

- wherein the submodule for spanning is able to traverse the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques *(see Paragraph [0033], "... the order of entity traversal during code generation could be random, or ordered by type, or sorted*

by name, or filtered so that only entities with a specific property are traversed, or the graph could be traversed in various other ways such as depth-first traversal.").

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Schloegel into the teaching of Goodwin to include wherein the submodule for spanning is able to traverse the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques. The modification would be obvious because one of ordinary skill in the art would be motivated to provide a framework under which it is possible to reason and/or prove qualities about the generated code *(see Schloegel – Paragraph [0034]).*

As per **Claim 50**, the rejection of **Claim 48** is incorporated; however, Goodwin and Barrett do not disclose:

- wherein the spanning step includes traversing the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques.

Schloegel discloses:

- wherein the spanning step includes traversing the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques *(see Paragraph [0033], "... the order of entity traversal during code generation could be random, or ordered by type, or sorted by name, or filtered so that only entities with a specific property are traversed, or the graph could be traversed in various other ways such as depth-first traversal.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Schloegel into the teaching of Goodwin to

include wherein the spanning step includes traversing the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques. The modification would be obvious because one of ordinary skill in the art would be motivated to provide a framework under which it is possible to reason and/or prove qualities about the generated code *(see Schloegel – Paragraph [0034])*.

10. **Claims 12, 34, and 53** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Goodwin** in view of **Barrett** as applied to Claims 1, 23, and 43 above, and further in view of **US 6,681,383 (hereinafter "Pastor")**.

As per **Claim 12**, the rejection of **Claim 1** is incorporated; however, Goodwin and Barrett do not disclose:

- after reconstructing the model at run-time, testing integrity of the reconstructed model.

Pastor discloses:

- after reconstructing the model at run-time, testing integrity of the reconstructed model *(see Column 21: 21-28, "Two procedures are used for Conceptual Model validation. For completeness, validation rules are implemented by directly checking the gathered data for the Conceptual Model, e.g., a class must have name, one attribute being its identifier and one service. For correctness, an extended formal specification language grammar is implemented in order to validate the syntax and meaning of all the formulas in the Conceptual Model.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Pastor into the teaching of Goodwin to include after reconstructing the model at run-time, testing integrity of the reconstructed model. The modification would be obvious because one of ordinary skill in the art would be motivated to ensure that the model is both correct and complete *(see Pastor – Column 20: 58-62).*

As per **Claim 34**, the rejection of **Claim 23** is incorporated; however, Goodwin and Barrett do not disclose:

- a submodule for testing integrity of the reconstructed model.

Pastor discloses:

- a submodule for testing integrity of the reconstructed model *(see Column 21: 21-28, "Two procedures are used for Conceptual Model validation. For completeness, validation rules are implemented by directly checking the gathered data for the Conceptual Model, e.g., a class must have name, one attribute being its identifier and one service. For correctness, an extended formal specification language grammar is implemented in order to validate the syntax and meaning of all the formulas in the Conceptual Model.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Pastor into the teaching of Goodwin to include a submodule for testing integrity of the reconstructed model. The modification would be obvious because one of ordinary skill in the art would be motivated to ensure that the model is both correct and complete *(see Pastor – Column 20: 58-62).*

As per **Claim 53**, the rejection of **Claim 43** is incorporated; however, Goodwin and Barrett do not disclose:

- after reconstructing the model, testing integrity of the reconstructed model.

Pastor discloses:

- after reconstructing the model, testing integrity of the reconstructed model *(see Column 21: 21-28, "Two procedures are used for Conceptual Model validation. For completeness, validation rules are implemented by directly checking the gathered data for the Conceptual Model, e.g., a class must have name, one attribute being its identifier and one service. For correctness, an extended formal specification language grammar is implemented in order to validate the syntax and meaning of all the formulas in the Conceptual Model.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Pastor into the teaching of Goodwin to include after reconstructing the model, testing integrity of the reconstructed model. The modification would be obvious because one of ordinary skill in the art would be motivated to ensure that the model is both correct and complete *(see Pastor – Column 20: 58-62)*.

11. **Claims 13, 14, 35, 36, 54, and 55** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Goodwin** in view of **Barrett** and **Pastor** as applied to Claims 12, 34, and 53 above, and further in view of **US 7,162,462 (hereinafter "Mutschler")**.

As per **Claim 13**, the rejection of **Claim 12** is incorporated; however, Goodwin, Barrett, and Pastor do not disclose:

- ensuring that all classes in the model belong to a common superclass.

Mutschler discloses:

- ensuring that all classes in the model belong to a common superclass *(see Column 5: 19-21, "The persistent object 210 represents a superclass from which the named object 220 and other persistent objects 222 are derived.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Mutschler into the teaching of Goodwin to include ensuring that all classes in the model belong to a common superclass. The modification would be obvious because one of ordinary skill in the art would be motivated to share data and methods relating to the classes' common dynamic behavior over time *(see Mutschler – Column 5: 40-43)*.

As per **Claim 14**, the rejection of **Claim 13** is incorporated; however, Goodwin, Barrett, and Pastor do not disclose:

- if all classes in the reconstructed model do not share a common superclass, automatically constructing a common superclass for those classes.

Mutschler discloses:

- if all classes in the reconstructed model do not share a common superclass, automatically constructing a common superclass for those classes *(see Column 5: 8-13, "... the effect of a common superclass can be achieved by adding structure fields or data areas representing the attribute data of the superclass to these structures or data blocks and providing procedures or functions in the rule engine program to access and manipulate them.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Mutschler into the teaching of Goodwin to include if all classes in the reconstructed model do not share a common superclass, automatically constructing a common superclass for those classes. The modification would be obvious because one of ordinary skill in the art would be motivated to share data and methods relating to the classes' common dynamic behavior over time *(see Mutschler – Column 5: 40-43)*.

As per **Claim 35**, the rejection of **Claim 34** is incorporated; however, Goodwin, Barrett, and Pastor do not disclose:

- a submodule for ensuring that all classes in the model belong to a common superclass.

Mutschler discloses:

- a submodule for ensuring that all classes in the model belong to a common superclass *(see Column 5: 19-21, "The persistent object 210 represents a superclass from which the named object 220 and other persistent objects 222 are derived.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Mutschler into the teaching of Goodwin to include a submodule for ensuring that all classes in the model belong to a common superclass. The modification would be obvious because one of ordinary skill in the art would be motivated to share data and methods relating to the classes' common dynamic behavior over time *(see Mutschler – Column 5: 40-43)*.

As per **Claim 36**, the rejection of **Claim 35** is incorporated; however, Goodwin, Barrett, and Pastor do not disclose:

- a submodule for automatically constructing a common superclass for those classes when all classes in the reconstructed model do not share a common superclass.

Mutschler discloses:

- a submodule for automatically constructing a common superclass for those classes when all classes in the reconstructed model do not share a common superclass *(see Column 5: 8-13, "... the effect of a common superclass can be achieved by adding structure fields or data areas representing the attribute data of the superclass to these structures or data blocks and providing procedures or functions in the rule engine program to access and manipulate them.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Mutschler into the teaching of Goodwin to include a submodule for automatically constructing a common superclass for those classes when all classes in the reconstructed model do not share a common superclass. The modification would be obvious because one of ordinary skill in the art would be motivated to share data and methods relating to the classes' common dynamic behavior over time *(see Mutschler – Column 5: 40-43)*.

As per **Claim 54**, the rejection of **Claim 53** is incorporated; however, Goodwin, Barrett, and Pastor do not disclose:

- ensuring that all classes in the model belong to a common superclass.

Mutschler discloses:

- ensuring that all classes in the model belong to a common superclass *(see Column 5: 19-21, "The persistent object 210 represents a superclass from which the named object 220 and other persistent objects 222 are derived.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of <u>Mutschler</u> into the teaching of <u>Goodwin</u> to include ensuring that all classes in the model belong to a common superclass. The modification would be obvious because one of ordinary skill in the art would be motivated to share data and methods relating to the classes' common dynamic behavior over time *(see <u>Mutschler</u> – Column 5: 40-43)*.

As per **Claim 55**, the rejection of **Claim 54** is incorporated; however, <u>Goodwin</u>, <u>Barrett</u>, and <u>Pastor</u> do not disclose:

- if all classes in the reconstructed model do not share a common superclass, automatically constructing a common superclass for those classes.

<u>Mutschler</u> discloses:

- if all classes in the reconstructed model do not share a common superclass, automatically constructing a common superclass for those classes *(see Column 5: 8-13, "... the effect of a common superclass can be achieved by adding structure fields or data areas representing the attribute data of the superclass to these structures or data blocks and providing procedures or functions in the rule engine program to access and manipulate them.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of <u>Mutschler</u> into the teaching of <u>Goodwin</u> to

include if all classes in the reconstructed model do not share a common superclass, automatically constructing a common superclass for those classes. The modification would be obvious because one of ordinary skill in the art would be motivated to share data and methods relating to the classes' common dynamic behavior over time *(see <u>Mutschler</u> – Column 5: 40-43)*.

Response to Arguments

12. Applicant's arguments with respect to Claims 7, 12, 29, 34, 48, and 53 have been considered but are moot in view of the new ground(s) of rejection.

In the remarks, Applicant argues that:

a) As noted above, Barrett's solution transforms a UML model into a meta model which it then saves as an XMI file. XML Metadata Interchange (XMI) is an Object Management Group (OMG) standard for exchanging meta data information via Extensible Markup Language (XML). It provides a mapping from UML (or another model) to XML. Importantly, this XMI information is stored in a separate file and Barrett makes no mention of representing the UML model within the source code itself. Applicant's claimed invention, in contrast, specifically calls for representing the model within the source code itself as source code and code attributes (Applicant's specification, paragraph [0061]).

Examiner's response:

a) Examiner disagrees. Applicant's arguments are not persuasive for at least the following reasons:

First, Barrett clearly discloses "representing the model within the source code itself, wherein the model is represented as source code and code attributes" *(see Column 5: 48-52, "In the requirements and specification phases 1 and 2 the system design is modelled. In the preferred embodiment, this model comprises a series of UML diagrams. The model is developed by the UML tool 15 and recorded in a file in an XMI/XML form."; Column 8: 24-45)*. Note that the UML model is saved as an XMI/XML file. Thus, the UML model is represented in source code in XMI/XML form. Barrett also clearly illustrates, in column 8, lines 30-45, that the UML model is represented as XML markup code containing various markup tags (source code) and tag attributes (code attributes).

Second, the claims only require representing the model in source code itself. This can be broadly interpreted as representing the model in source code. The claims are not limited to the scope of representing the model in the source code for the application. Appellant is reminded that in order for such limitations to be considered, the claim language requires to specifically recite such limitations in the claims, otherwise broadest reasonable interpretations of the broadly claimed limitations are deemed to be proper.

In the remarks, Applicant argues that:

b) Another difference between Applicant's claimed invention and Barrett's solution is that Barrett reconstructs the runtime model from this stored XMI-representation and not from a model represented in code of the executable application itself. As noted above, Barrett's solution "maintains the meta-model associated with the executing system" (Barrett, col. 5, lines 37-38). Moreover, the meta-model remains available to the UML tool and changes to the model have

immediate effect on the executing system by addition, removal and/or replacement of system components (Barrett, col. 5, lines 39-44). These features make it obvious that Barrett's system does not reconstruct the model from compiled code since it can be changed in the UML tool while the system is running. Thus, it is clear that the model (in Barrett) is not actually reconstructed from the executable application, but rather from a "meta-model associated with the executing system" stored in a separate file.

Examiner's response:

b) Examiner disagrees. Barrett clearly discloses reconstructing the model from the executable application while the executable application is running *(see Column 5: 37-44, "During execution of a system, the RAS 10 maintains the meta-model associated with the executing system. During this phase, the meta model remains accessible to the UML tool. Changes made to the meta model have immediate effect on the executing system by addition, removal and/or replacement of components, and changes to the structure of the bindings between instances according to changes to the meta model.")*. Note that the meta-model is associated with the executing system (executable application). Thus, the changes made to the meta-model (reconstructing the model) occur during system runtime.

Conclusion

13. Any inquiry concerning this communication or earlier communications from the Examiner should be directed to Qing Chen whose telephone number is 571-270-1071. The

Examiner can normally be reached on Monday through Thursday from 7:30 AM to 4:00 PM. The Examiner can also be reached on alternate Fridays.

If attempts to reach the Examiner by telephone are unsuccessful, the Examiner's supervisor, Wei Zhen, can be reached on 571-272-3708. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the TC 2100 Group receptionist whose telephone number is 571-272-2100.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

/QC/
June 26, 2008

/Wei Zhen/

Supervisory Patent Examiner, Art Unit 2191